

**A NEIGHBORHOOD LOCKING TECHNIQUE FOR
INCREASING CONCURRENCY AMONG TRANSACTIONS**

FIELD OF INVENTION

The invention relates to the field of data processing systems. More specifically, the invention relates to the locking techniques for management of data in database systems and modes of locks acquired on various data items.

BACKGROUND OF INVENTION

A database management system (DBMS) supports transactions to access data items stored in a database. Before actually accessing the items, a transaction acquires locks on the data items that it wishes to access. Typically, a read only or read/write access may be requested on the data item. Generally, this is termed as shared and exclusive access in the literature. Thus locks are requested in a pair <resource, lock mode>, where the lock mode indicates the kind of access the transaction needs on the resource. Typically these modes are denoted as S or X lock modes. If a transaction owns an X lock on a given resource, no other transaction can own any lock in any mode on that resource. This is generally known as a lock conflict. Thus X conflicts with all other modes, and S conflicts with X mode. However, S is compatible with another S mode lock on the resource.

In the interest of performance, often it is useful to define structure on data. For example, data in a typical database can be classified into tables and records in a table. An X lock on a table can be considered as an X lock on all rows in the table. This technique is termed as hierarchical locking. Only using X and S locks are inadequate in supporting hierarchical locking, and therefore a typical DBMS supports many more lock modes, to enhance transaction concurrency.

For example, most DBMSs use IS, S, IX, X, SIX and U lock modes for controlling concurrent accesses to data items. The IS mode refers to intention shared (obtained on the table), intention to take S locks on underlying rows. The IX mode refers to intention exclusive (obtained on the table), intention to take X locks on underlying rows. The S mode refers to shared, if obtained on the table, implies an S lock on all underlying rows. The SIX mode refers to shared intention exclusive, a combination of S and IX, obtained on the table.

The U mode refers to update lock (can be obtained on the table), it means that a X lock may requested on the data item in the future.

Typically a database consists of isolated transactions. The degree of isolation may vary among transactions. The American National Standards Institute (ANSI) has defined various degrees of isolation from strict SERIALIZABLE to DIRTY READ. Existing lock modes may be sufficient to synchronize data access among serializable transactions. But, the lock modes described above are too restrictive in terms of avoidable conflicts as per the semantics of non-serializable transactions.

SUMMARY OF THE INVENTION

New lock modes are based on the concept of neighborhoods and are applied to spaces in indexes. The new lock modes include a Read, or shared, neighborhood (Snei) lock mode and a write neighborhood (Xnei) that enhance concurrency among non-serializable transactions.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a diagram of a database management system that supports neighborhood lock modes.

FIG. 2 is a schematic diagram showing locking operations in the database management system.

FIG. 3 is a schematic diagram showing a neighborhood associated with a tuple.

FIG. 4 is a schematic diagram showing how a write operation inserts a tuple into the neighborhood.

FIG. 5 is a schematic diagram showing how two transactions operate concurrently with respect to the neighborhood.

FIG. 6 is a schematic diagram showing how the two transactions can both operate concurrently during an insert operation.

FIG. 7 is a table showing how neighborhood lock modes operate with respect to other lock modes.

FIG. 8 is a schematic diagram showing how a read operation operates concurrently with a neighborhood insert operation.

DETAILED DESCRIPTION

FIG. 1 shows a database system that includes database 8 and a processor 4 (Central Processing Unit (CPU) and memory) that stores and executes software for a Database Management System (DBMS) 6 and a lock manager 10. A computing device 2 containing processor 4 in one example is a server. The processor 4 may be a single CPU or multiple CPUs that operate in the server 2. The database 8 is typically stored on a plurality of Direct Access Storage Devices (DASDs) although only one such device is illustrated in FIG. 1. The DASDs may be, for example, disk packs, and it will be understood by those skilled in the art that the database system includes the required DASD controllers and channels that establish the communication between the database 8 and the processor 4. In another example the database (8) may be stored inside the main memory (4) itself.

Terminals 1 are any Input/Output (I/O) device that accepts requests from and returning results to users. For example, the terminals may be Personal Computers (PCs) or applications that use the advertised application programmer interface (API). Transactions and queries are submitted from the terminals 1 and are processed concurrently by the DBMS 6.

A Lock Manager (LM) 10 is a module of the DBMS 6 that helps the transactions in acquiring and releasing the locks on data items in the database 8 in a controlled manner. The LM 10 needs to make sure that conflicting accesses to a given data item are not granted simultaneously. For example, if a simultaneous read and write access has been granted on a data item, then the reader of that data item may see an incomprehensible state of the data item. Thus some lock requests may not be granted to different transactions simultaneously, since there can be conflicts in the lock grant requests.

All transactions request that locks be granted in a desired lock mode. For example, the reader may request a lock in S mode and a writer may request the same lock in X mode. If S and X are defined not be compatible, then the LM 10 guarantees that no writer has access permission. In other words, no X lock exists on the data item if a S lock has already been obtained on that data item. Thus a reader can read a consistent state of the data item after S lock has been obtained on that data item. The LM 10 uses these and other conventional lock modes along with new neighborhood based lock modes described below.

Referring to FIG. 2, the triangle represents a tree index 12. A range of values in the index 12 is locked by always ensuring that the next tuple via the index 12 is locked at the end of select, insert and delete operations. Consider an example of the usage of the aforementioned lock modes in the case of an insert operation 14 and a read operation 17, both

executing under a non-serializable isolation mode. Additionally, assume that the read operation 17 uses the tree index 12. Also assume that the DBMS 6 (FIG. 1) is designed to support serializable scans when using the tree index 12.

The items A and C represent tuples in a table 16, on which the index 12 is defined.

- 5 Assume that the read operation 17 has already visited A and C. The insert operation 14 needs to insert a new tuple B that is positioned in the index 12 between A and C. At this point, before allowing the insert operation 14 to insert tuple B in the index 12, the lock manager 10 must ensure that no serializable scanner (the read operation 17) has looked at the surrounding area that includes tuples A and C. Otherwise the read operation 17 will be non repeatable
10 even for serializable transactions.

- Consider that the read operation 17 holds a lock on tuple C in "S" lock mode. At this point the insert operation 14 needs to lock what will be the tuple next to tuple B in the index 12, i.e. tuple C. The insert operation 14 checks if a read operation under serializable isolation has read tuple C or not. If the insert operation 14 can request an X lock on tuple C, for
15 instant duration, inserts in the same area could be concurrent. But as can will be seen from the compatibility matrix in FIG. 7, the lock manager 10 will not allow the insert operation 14 to lock on tuple C in mode "X" until the "S" lock on tuple C is released by read operation 17.

- New lock modes Snei and Xnei are recognized by the lock manager 10 and allow the insert operation 14 to succeed when there is no serializable reader in the vicinity. The Snei
20 lock mode refers to a read on the neighborhood. The Xnei lock mode refers to write permission on the neighborhood. In one embodiment, these two lock modes may be used for tuples accessed by non-serializable transactions. Serializable transactions, on the other hand, hold these locks implicitly since they hold stronger locks on the tuple.

- For example, the read operation 17 may be a non serializable read operation in the
25 mode Sn, which stands for S-non-serializable. The insert operation 14 then requests a Xnei lock mode through lock manager 10, which is compatible with Sn, and therefore granted. Thus the insert operation 14 allowed to succeed by the lock manager 10 even in the presence of concurrent non-serializable read operation 17 in the same vicinity.

- Let $(V, <)$ be a totally ordered set. Let v be an element of V . By the left neighborhood
30 of v in V , we mean the subset $\{v' \mid v' < v\}$ of V . Suppose we are given an ordered sequence v_1, v_2, v_3, \dots of elements from V . By the least left neighborhood of element v_i relative to the given ordered sequence we mean the subset $\{v \mid v_{i-1} < v < v_i\}$ of the left neighborhood of v_i . This set represents the 'gap or space' in the given sequence, that is, all the elements in the set

can be inserted between v_{i-1} and v_i , and no more elements from V can be inserted there without violating the given ordering.

Referring to FIG. 3, each table 15 stores tuples from a predefined countable domain D of tuples. All tuples can be ordered in domain D in many different ways. For a database system, these orderings are generally determined by ordered indices defined on the table 15. There is also a natural physical ordering for each table 15. The left neighborhood for a tuple relative to the physical ordering consists of all free cells that lie before the tuple in the corresponding table 15. For example, the neighborhood for tuple G in FIG. 3 are the free cells 18 that reside before tuple G but after tuple A .

Only a subset of domain D will be present in table 15. The tuple G will have different least left neighborhoods relative to the total orderings defined by the index 12 and the physical ordering on table 15. The neighborhood of tuple G is defined as the union of all its least left neighborhoods 18. The neighborhood 18 can be visualized as the corresponding gaps to the left of tuple G in all index trees 12 and the physical representation of the table 15.

A tuple G and its neighborhood 18 are two different, though tightly related, entities. Transactions distinguish between access permission that they have on the tuple G and access permission on the neighborhood 18. A transaction may be allowed to read and/or write the neighborhood 18 of tuple G , even when it has no access permissions on the tuple G .

Referring to FIG. 4, a write operation 24 on the neighborhood 18 splits the neighborhood 18 into two neighborhoods 20 and 22 by inserting a new tuple B in the original neighborhood 18. A read operation of a neighborhood, is a fictitious reading of a fictitious tuple in the neighborhood.

The Snei and Xnei lock modes are compared with read and write operations that use eleven lock modes: IS, S, IX, X, SIX, U, Sn, Xn, Un, Xnei and Snei. The Sn, Xn, and Un lock modes are the counter parts of S, X and U, but are only used by non-serializable transactions. The S, X, U, Sn, Xn, Un, Xnei and Snei modes are used to lock tuples and their neighborhoods, and IS, S, IX, X, SIX and U modes are used to lock tables. The semantics of these lock modes are described below.

Referring to FIG. 5, suppose a transaction T holds a lock on a data item d . What transaction T can do with data item d and/or its neighborhood 30 (if d is a tuple) is explained below. Also, discussed below is what another transaction T' can or cannot do with data item d and/or its associated neighborhood 30. Data items contained in a course granule are referred to as ingredients.

S – Shared: transaction T locks data item d and the neighborhood 30 in shared mode. Any transactions changing data item d or the neighborhood 30 are prohibited. Transaction T can read data item d and its ingredients, but it cannot write them. Transaction T also cannot modify the data neighborhood. A second transaction T' can read data item d and its ingredients. Transaction T' cannot write data item d or its ingredients, nor modify the data neighborhood 30.

IS – Intention shared: Transaction T is planning to read some lower level data item (row) of the table 32. Transaction T cannot read the lower level portion of table 32 until it obtains a lock on the related data item d.

X – Exclusive: Transaction T locks data item d and the neighborhood 30 in exclusive mode. No other transactions are allowed to be active on data item d or the neighborhood 30, not even in Snei mode. Transaction T can read and write data item d and its ingredients. Transaction T can also modify the neighborhood 30. Transaction T' cannot read or write data item d or its ingredients, nor modify the data neighborhood 30.

IX – Intention exclusive: Transaction T is planning to write some lower level data item (row) of the table 32. Transaction T cannot read or write any row in the table 32 until proper locks are obtained on the row.

SIX – Shared intention exclusive: Transaction T requests this lock mode on table 32 when transaction T needs to read many rows on table 32 but write only a few rows.

Transaction T can read rows of table 32, but it cannot write any row of table 32 without a proper lock on the row. Transaction T' can only read rows of table 32 after obtaining proper locks.

U – Update: A transaction that reads a data item but expects to write the same data item later locks the data in U mode to convert it later to X mode if it writes or to S mode if it decides not to write the data. This mode helps in avoiding deadlocks in converting S locks to X locks. Transaction T can read, but not write, d and its ingredients and cannot modify the neighborhood 30 if data item d is a tuple. Transaction T' cannot obtain a new read/write permission for data item d or its ingredients. However, transaction T' can read data item d and its ingredients if it has already got permission to do so. It cannot write data item d or any ingredients, nor modify the neighborhood 30.

Referring to FIG. 6, the following modes are used to promote a higher level of concurrency for non-serializable isolation levels, especially for scanning tables: physical, hash, or a tree index. These modes are valid for locking tuples. In FIG. 6, data items A and C represent pointers in an index 40 that point to associated tuples A and C, respectively.

Data Item A is the next reserved data item to the left of data item C. The neighborhood 42 for tuple C is the free space in a table (not shown) between tuples A and C.

Sn – Non-serializable shared: Transaction T locks tuple C in shared mode, but does not lock its neighborhood 42. Transaction T can read tuple C, but cannot write it.

- 5 Transaction T cannot modify the tuple neighborhood 42. Transaction T' can read tuple C, but cannot write it. However, transaction T' can modify the tuple neighborhood.

- 10 Xn – Weak exclusive: Transaction T locks tuple C in exclusive mode, but does not lock its neighborhood 42. Transaction T can read and write tuple C but cannot modify the tuple neighborhood 42. Transaction T' cannot read or write tuple C but can modify the tuple neighborhood 42.

- Xnei – Weak Exclusive neighborhood lock: Transaction T does not want to read or write tuple C. Transaction T locks the neighborhood 42 for shrinking by inserting a new tuple B. Transaction T cannot read or write tuple C but can shrink the tuple neighborhood 42. Transaction T' can read and write tuple C and can also shrink the tuple neighborhood 42. Note that transaction T' cannot have an X lock on tuple C. This means that transaction T' cannot have exclusive rights over the neighborhood 42.

- 20 Un – Non-serializable update: Transaction T locks tuple C in shared mode, but expects to write it soon. Transaction T does not have interest in the tuple neighborhood 42. Transaction T can read tuple C, but cannot write it and cannot modify the tuple neighborhood 42. Transaction T' cannot obtain a new read/write permission for tuple C. However, transaction T' can read tuple C if it has already got permission to do so, but it cannot write tuple C. Transaction T' can also modify the tuple neighborhood 42.

- 25 Snei – Weak Shared neighborhood lock: This operation is an indication that transaction T has finished an ongoing orderly scan. The lock acts as a sentry for transaction T. Transaction T cannot read or write tuple C and cannot modify the tuple neighborhood 42. Transaction T' can read and write tuple C and can modify the data neighborhood 42. Transaction T cannot have an X lock on tuple C.

- 30 A lock compatibility matrix is shown in FIG. 7 for some of the various lock modes described above. Each row defines what happens when a new lock request is made on the same resource with the given lock mode, and a previous lock exists on the same resource in the mode indicated by the columns. An entry of one means that the new lock mode can be granted, given the existing resource lock. A value of zero means that the new lock cannot be granted. A lock request can be granted only if it is compatible with all locks that are currently held on the given resource. In one example, the matrix shown in FIG. 7 is stored in

memory somewhere in the server 2 (FIG. 1). The lock manager 10 (FIG. 1) is operated by the processor 4 and uses the matrix to determine when to grant lock modes to different transactions.

One example is described in FIG. 8. If a read operation 50 on tuple C is non-serializable, then it will hold a Sn lock on tuple C. An insert operation 52 wants to insert tuple B and needs to lock the neighborhood of tuple B. The insert operation 52 requests and is granted a Xnei lock on tuple C. In the non-serializable mode, the insert operation 52 and the read operation 50 will be concurrent without any lock conflict. In other words, the Sn lock on tuple C and the Xnei lock on tuple C exist concurrently.

On the other hand, if the read operation 50 is a serializable reader, it will hold a S lock on tuple C. Since Xnei is not compatible with the serializable S lock, the insert operation 52 will have to wait for the serializable read operation 50 to commit or rollback. This maintains serializability for the read operation 50, by maintaining the repeatable read semantics for the serializable operation.

The lock modes described above are superior to other lock modes. Assume two non-serializable operations delete separate rows B and C such that row B is next to row C, and row C is next to row D in a non-unique index. In other DBMSs, the delete for row B takes an X lock on row B and a X (or other proprietary weaker locking modes e.g. NX in IBM's DB2) lock on row C. At this point the delete for row C cannot proceed since it needs a X lock (or at least a W lock in IBM's DB2), on row C. In the neighborhood locking scheme, the delete for row B can take a Xw lock on row B and a Xnei lock on row C. The delete for row C takes a Xw lock on row C, which is compatible and an Xnei lock on row C. Thus both deletes are concurrent.

All database operations can be reduced to four basic tuple operations. A transaction may request a read of a given tuple, an insert of a new tuple, a delete of a given tuple, and an update of a given tuple. A read operation typically reads a range via an index. Since the neighborhood between the current tuple and the next tuple should be locked to ensure the completion of the read operation, a Snei lock may be requested on the next tuple in the index for an index based read operation. For a table scan, without an index, a Snei lock may be requested on a fictitious end of table tuple.

An insert operation may result in multiple index insertions for the new tuple. For each index insertion the transaction requests a Xnei lock on the neighborhood between the tuple that will be inserted and the existing next tuple on that index. A delete operation requests a similar Xnei lock on the neighborhood between the tuple to be deleted and the next

tuple. An update operation consists of index deletes followed by an index update. Thus an update operation consists of two Xnei requests for each effected index. Index deletes executing under serializable isolation may ask for a more restrictive lock mode, such as X, to ensure that no insertion or deletion can happen in the neighborhood.

5 Using tuple neighborhoods helps in enunciating a clear design for various database actions like read, insert, delete and update. Additionally, lock modes designed based on the concept of tuple neighborhoods provide better concurrency among transactions, as per the semantics of that transaction's isolation level, than previously available.

The system described above can use dedicated processor systems, micro controllers, programmable logic devices, or microprocessors that perform some or all of the operations. Some of the operations described above may be implemented in software and other operations may be implemented in hardware.

For the sake of convenience, the operations are described as various interconnected functional blocks or distinct software modules. This is not necessary, however, and there may be cases where these functional blocks or modules are equivalently aggregated into a single logic device, program or operation with unclear boundaries. In any event, the functional blocks and software modules or features of the flexible interface can be implemented by themselves, or in combination with other operations in either hardware or software.

10 Having described and illustrated the principles of the invention in a preferred embodiment thereof, it should be apparent that the invention may be modified in arrangement and detail without departing from such principles. Claims are made to all modifications and variation coming within the spirit and scope of the following claims.